Nanotechnology **19** (2008) 115708 (17pp) doi:10.1088/0957-4484/19/11/115708

Fault-tolerant sub-lithographic design with rollback recovery

Helia Naeimi[1] **and André DeHon**[2]

[1] Department of Computer Science, California Institute of Technology, Pasadena, CA 91125, USA
[2] Department of Electrical and System Engineering, University of Pennsylvania, 200 S. 33rd Street, Philadelphia, PA 19104, USA

E-mail: helia@caltech.edu

Received 14 June 2007, in final form 5 September 2007
Published 19 February 2008
Online at stacks.iop.org/Nano/19/115708

Abstract
Shrinking feature sizes and energy levels coupled with high clock rates and decreasing node capacitance lead us into a regime where transient errors in logic cannot be ignored. Consequently, several recent studies have focused on feed-forward spatial redundancy techniques to combat these high transient fault rates. To complement these studies, we analyze fine-grained rollback techniques and show that they can offer lower spatial redundancy factors with no significant impact on system performance for fault rates up to one fault per device per ten million cycles of operation ($P_f = 10^{-7}$) in systems with 10^{12} susceptible devices. Further, we concretely demonstrate these claims on nanowire-based programmable logic arrays. Despite expensive rollback buffers and general-purpose, conservative analysis, we show the area overhead factor of our technique is roughly an order of magnitude lower than a gate level feed-forward redundancy scheme.

1. Introduction

Shrinking feature sizes make our components more susceptible to transient faults for two reasons:

(i) The fault rate of each individual device increases. That is, feature size scaling and voltage level reduction shrinks the amount of critical charges holding logical state on each node; this in turn makes each node more susceptible to transient faults, e.g. an ionized particle strike has a higher likelihood of being fatal as the critical charge is reduced in a node [1].
(ii) The number of devices we can place per chip increases with shrinking feature size; consequently, each chip packs more devices which may fail.

At the chip level, fault rate increases approximately as the product of these two effects (see equation (3) and associated text). Consequently, fault-tolerant design approaches will soon become an inevitable part of system design.

Fault-tolerant approaches must detect or correct transient errors in the system. Error detection or correction requires some form of information redundancy, which usually results in additional area overhead in the system.

Minimizing area overhead at a given reliability level is the primary goal and the key metric in this field; this optimization becomes more challenging as the device fault rate increases. In this paper we show that by exploiting a *rollback recovery* approach [2] we can design reliable nanotechnology systems that have close to a factor of six lower area compared to the previous fault-tolerant nanotechnology designs that were based on von Neumann's *feed-forward* recovery scheme [3, 4].

Rollback recovery techniques are a well-known class of fault-tolerant design strategies. Generally in rollback recovery techniques, errors are *detected* with *spatial* redundancy (e.g. a duplicated copy of the logic) and *corrected* with *temporal* redundancy (e.g. repeating the operation). The system runs at high speed when there are no errors, but when an error is detected, the system stops and repeats the affected operation to generate the correct result. *Rollback recovery* schemes exploit the fact that most of the operation cycles pass with no error occurrence, and therefore the recovery process occurs infrequently and the throughput impact is potentially low. In contrast, *feed-forward recovery* schemes provide enough *spatial* redundancy in the system to *detect* and *correct* errors with no *temporal* redundancy (e.g. voting among three copies of logic).

The key advantage of rollback recovery schemes is lower area overhead compared to feed-forward recovery schemes. This is already clear with the simplest examples of rollback and feed-forward recovery techniques. The *triple modular redundancy with a voter* (a feed-forward recovery scheme) takes roughly 3 times the area of the unprotected design, while the *duplication with comparison* system (a rollback recovery) takes only 2 times the area. The *triple modular redundancy* and *duplication with comparison* systems are only adequate when the fault rate is sufficiently low, i.e. they correct or detect a single error in the system. As fault rates and reliability goals increase, the gap between these two techniques widens, as we quantify in this paper.

There are fundamental reasons for the *rollback recovery* scheme to be more area efficient than *feed-forward* schemes. When transient faults do not occur on most of the cycles, the spatial redundancy in the feed-forward recovery is not used most of the time; therefore the large area allocated for correction is effectively wasted on most of the cycles. On the other hand the time redundancy used for correction in the rollback recovery system is spent only when it is needed, i.e. an error is detected. This allows the *rollback* scheme to be more efficient in both area and the area–time product.

Despite the absolute advantage of the traditional rollback recovery in area overhead, there is a potential throughput drop in the rollback approach used for nanotechnology systems if it is not designed properly. That is, if rollback occurs too frequently, it can have a significant, detrimental impact on throughput. In traditional systems with low device fault rate and smaller system size, rollback frequency is guaranteed to be low. However, in nanoscale systems, the fault rate will be much higher and the system size will be much larger; both effects increase the rollback frequency and therefore can severely impact the system performance. To make the rollback technique work for nanotechnology devices, we explore *fine-grained rollback*. In this technique we partition the system into small blocks and apply rollback recovery on each block independently. If the block's size is small enough to guarantee infrequent error occurrence and consequently infrequent rollback operation in each block, then high system performance is achievable. To optimize the performance further, we implement *streaming buffers* between the blocks. This allows blocks to operate independently reducing the impact of each error. As demonstrated later in this paper (sections 3.3 and 5), these techniques help maintain high system performance for a wide range of fault rates.

To further keep overhead low and minimize the complexity of the rollback process, we exploit devices with different reliability factors. For example, reliable controllers, which take a tiny fraction of the system area (see section 4.3), are implemented with coarser, more reliable devices, while the rest of the circuitry is implemented with smaller but less reliable devices. Since only a tiny fraction of the system uses coarse devices, their use has negligible impact on the design area. This same strategy is also used in some of the feed-forward schemes [3–5].

To demonstrate the benefit of the *fine-grained rollback* technique, we develop a full area and reliability estimate for the rollback recovery technique and ground a detailed area and reliability analysis in a specific nanotechnology architecture model of the *nanoPLA* [6]. We also introduce a novel efficient multi-way comparator design, optimized for a *nanoPLA* architecture model, or any other two-level implementation.

The main contribution of this paper is the broad comparison between rollback and feed-forward recovery across a wide range of fault rates. In order to perform this broad comparison, we have developed novel variations on rollback recovery. Our contributions further include: (a) redesigning the rollback recovery technique such that we can push this technique below the coarse building block level and toward the gate level, (b) treating block size as a parameter for engineering optimization and quantifying its impact on performance and area, (c) separating detection block size from rollback block size and independently optimizing these parameters, and (d) quantitatively assessing the choice of parameters and their performance impact across a wide range of fault rates.

The rest of this paper is organized as follows: section 2 reviews sources of transient faults and recent fault-tolerant designs for nanotechnology devices. The details of our fine-grained rollback design are developed in section 3. In section 4, we show how this design can be implemented with the nanoPLA architecture model. In section 5, we compute the reliability of the system and show that for a high reliability goal (*failure in time*, FIT, of 360) the redundancy of our technique is much lower than a feed-forward recovery scheme. Section 6 provides the complete area estimation results, including performance simulations of the system to estimate the throughput impact. The conclusion comes in section 7.

2. Background

2.1. Transient fault sources

Many different sources can give rise to transient faults including: high energy ionized particle impacts, thermal noise, shot noise and power supply noise. Advanced VLSI systems with lower supply voltages and higher system integration (i.e. integrating more devices which may fail) increase the probability that any of the above sources disrupts logic. Feature size and voltage scaling lead to small node capacitance and voltage, resulting in decreased critical charge on nodes holding logical states. With fewer electrons representing states, each node in the system becomes more susceptible to charge disruption. For example:

(i) High energy ionized particles, such as alpha particles, disrupt logic by removing the critical charges at a node. Not all the alpha particle hits are fatal, but as the critical charge reduces the probability that an alpha particle hit becomes fatal increases. It has been shown that alpha-particle-induced transient fault rates per chip increase 30 times as the manufacturing process goes from 0.25 to 0.18 μm and the supply voltage drops from 2 to 1.6 V; at the same time the transient fault rates due to the neutron's impact increased by 20% [1].

(ii) As we increase the clock frequency and further reduce supply voltage, shot noise becomes a significant source of transient faults [7]. Fault rate due to shot noise increases as the ON-state current decreases. The ON-state current relates to the node charges as below:

$$I \approx Q/\Delta T \tag{1}$$

where Q is the critical charge on a node which decreases as feature size and supply voltage scale down and ΔT is the clock period for charging the node. Kish [7] concludes that, if we want to avoid transient faults from shot noise while operating with limited power density (e.g. ITRS 2005 maximum tolerable power dissipation estimate of 250 W cm^{-2} [8]) we cannot run highly integrated chips at high clock rates. Alternately, if we do want to run chips with 10^{12} devices at 1 GHz or higher, Kish's equations suggest we should be prepared for fault rates in excess of $P_f = 10^{-20}$. Note that these lower bound calculations make assumptions only about the available current density and clock rate and not about any properties of a particular technology.

(iii) As we decrease voltage and charge at a node, we also increase the probability that random thermal noise can disrupt a node (e.g. [9, 10]). When energy minimization is a premium (e.g. battery application), we can reduce voltage at the expense of higher fault rates (e.g. [11]). Energy minimization is also becoming a premium in high performance, highly integrated devices. As noted above, power densities are limited for practical cooling. Kish [12] estimates that these practical power density limits prevent standard (e.g. ITRS/Moore's law) performance and density scaling from continuing much longer without introducing high fault rates ($P_f = 10^{-20}$–10^{-5}) due to thermal noise.

Note that the effective transient fault rate is a combination of these and other phenomena. We have reviewed this set for illustration purposes. While we cannot state definitively the fault rate associated with a particular technology, it is clear that we will see significantly increased fault rates. In our analysis here, we characterize solutions as a function of fault rate so that our results will be broadly useful as the community develops better estimates for the fault rates of particular technologies.

2.2. Feature size scaling

One of the most important challenges to scaling feature sizes is the cost of the necessary fabrication process. Sub-lithographic, bottom-up synthesis techniques may offer an economical alternate to costly lithographic feature size scaling. Molecular-scale electronic elements like *nanowires* (e.g. [13, 14]), which are only a few atoms wide and microns [15] to millimeters [16] long, have been successfully constructed in chemistry labs. These new sub-lithographic technologies with 10 nm full pitch semiconducting and metallic nanowires [17] may enable terascale system integration. With selective NiSi conversion of the nanowire, 10 μm long nanowires can have resistances in the hundreds of kΩ to M Ω region [17, 18]. In addition to nanowires which provide very high interconnect density, sub-lithographic electronic devices have been demonstrated that enable computation at the same small dimensions, including reconfigurable molecules [19], which provide reconfigurable switches, and doping techniques [20–22], which enable gate-controlled junctions. Using the above devices we can design reconfigurable or restorative nanowire crossbars.

One promising proposed architecture model built upon these nanoscale building blocks is the *nanoPLA* [18] which is an interconnected nanowire crossbar. Each building block in the nanoPLA model consists of two reconfigurable crossbars and two restorative crossbars built from the building blocks highlighted above. Each of the nanoPLA building blocks has functionality similar to a single PLA plane. Section 4 provides more details on the structure of the nanoPLA.

2.3. Failure in time

A widely used metric for the reliability of a fault-tolerant design is the average number of failures seen per one billion hours of operation; this is known as the number of 'failures in time' or the FIT rate. The system will see device upsets continuously: however, as long as the system properly detects these upsets and prevents them from propagating into the computation, the computation remains fault-free. As a result, the system runs correctly until it fails to detect a set of device upsets and allows them to propagate errors into the computation. Modern reliable systems demand FIT rates between a hundred and a thousand.

Another system reliability metric is the per cycle system failure probability. The failure rate of a system is the probability that an undetected error strikes the system on a cycle, $P_{\text{sys\_und\_err}}$. FIT and system failure probability are related through the system clock speed. The system failure probability is the product of the FIT rate and the number of cycles in 10^9 h:

$$P_{\text{sys\_und\_err}} = \text{FIT} \times 3600\ \text{s h}^{-1} \times 10^9\ \text{h} \times \text{Frequency}.$$

For example, in a system with FIT = '360' and system frequency of 10 GHz, the undetected error probability of the system is 10^{-20}. Since the FIT rate of 360 and system frequency of 10 GHz are plausible assumptions for future system generation, we use this minimum system failure rate ($P_{\text{sys\_und\_err}} = 10^{-20}$) in the simulations in this paper and also recalculate the analysis of the techniques in previous work for this failure rate to compare results.

2.4. Previous work on fault-tolerant nanotechnology designs

Recent fault-tolerant techniques that address high fault rates and high system integration for nanotechnology designs mainly employed *feed-forward* recovery techniques [3, 4]. In *feed-forward* recovery the designer provides adequate spatial redundancy in the system such that the errors will be detected and corrected with no interruption in the computation.

The common, fine-grained *feed-forward* fault-tolerant techniques for nanotechnology designs are based on *multiplexing* the logic gates, which was originally developed by

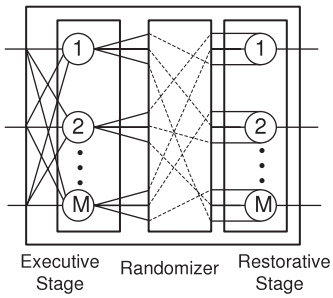

Figure 1. This figure shows a multiplexed unit to implement a reliable unit using von Neumann multiplexing technique.

von Neumann as *nand-multiplexing* in 1956 [5]. In the *multiplexing* technique, reliability is achieved by logic replication. Each bit is replicated M times and represented by the bundle of M wires. Computations are also replicated M times. In the majority multiplexing version, majority voting corrects errors in the logic. To prevent the voters from becoming a single point of failure, the voters are replicated as well. The trick is to make sure that a stage of computation and voting reduces the number of errors which exist in the bundle of wires which represent each bit.

For the multiplexing scheme, each processing unit is replaced by replicated copies of the processing unit and restoring module (NAND or MAJORITY gates). Each of the M wires of an input bundle has a separate and independent path through the multiplexed unit. A multiplexed unit consists of two stages, each using M processing units (see figure 1). The first stage is the executive stage which performs the actual logic operation and generates replicated results of the logic function. The second stage is the restorative stage. The restorative stage performs the redundant voting on the output of the executive stage and is responsible for improving the output reliability. The executive stage is connected to the restorative stage through a randomized interconnect; this randomization improves the reliability of the design by guaranteeing errors arriving at the restoration stage are statistically independent (figure 1). In recent work [4], it is shown that *majority* gates perform better than NAND gates, resulting in more compact, fault-tolerant designs. All the devices in the first and second levels and the randomized interconnects fail with equal probability. The total area overhead of this design is lower-bounded by its replication factor. The replication factor of this design is $2 \times M$. It is shown in [4] that majority multiplexing can be further optimized by sharing one restoration stage among multiple executive stages. Let L be the number of executive stages that share a restorative stage. The value of L impacts the reliability of the system, and there is a lower bound on it based on the desired system reliability. For a system with M multiplexing factor and L executive stages for one restoration stage, the replication factor is $((L+1)/L) \times M$. Note that the total area overhead is larger than the replication factor when considering the wiring area required by the randomized interconnects, particularly when M is large.

3. Design structure

Rollback recovery has been widely used for large block sizes with coarse-grained recovery, typically at the processor level [23–25]. In this section we design a *fine-grained* rollback technique that can tolerate higher fault rates than previous rollback techniques and achieve high system reliability. Later in this section, we show how the block size affects the reliable system design and why small blocks (i.e. at logic-level size) are essential.

This fine-grained rollback design has a two-level hierarchical structure, as shown in figure 2. At the base, the system is partitioned into fine-grained blocks called *detection blocks*. Each detection block has an embedded fault detection circuit to guarantee detection of a certain number of errors inside the block. At the next level the detection blocks are clustered to form a *rollback* (or *RB* for short) block. Each RB block guarantees the correctness of its output signals by performing rollback operations. Once a detection block inside an RB block signals an error, all the blocks inside the RB block stop their normal processes and the RB block *rolls back*, meaning it returns to a previously error-free state, recovers the inputs which arrived subsequent to that state and repeats the affected operations to generate the correct result.

The interconnects between the RB blocks are *buffered connections* that are designed to facilitate relatively independent operation flow between the RB blocks, i.e. the buffered connection provides buffer capacity between RB blocks, allowing an RB block to continue while an adjacent RB block is in rollback mode. Buffered connections are natural for many streaming systems and have often been used for large-scale concurrent computations (e.g. [26–31]).

The above building blocks—*detection block*, *RB block* and *buffered connections*—are developed in detail in the rest of this section.

3.1. Detection block

The detection block consists of the logic circuit block protected with enough redundant data to make errors in the logic circuit identifiable. A checker circuit follows the original circuit block and the redundant logic circuitry to detect any error at the output signals of the logic circuits. The main idea behind error detection is to compute redundant data *concurrently* with the main computation and compare the main and the redundant output signals, detecting any error in the main computation (figure 3). There are many different ways to generate the extra information to protect the main block [32] (e.g. parity signals [33], error correcting codes [34] and logic replication [35]).

Here we use a simple error detection technique, *replication with comparison*. It consists of multiple (R) independent copies of the main logic block, followed by a checker, which detects any disagreement among the copies of the logic block. We select R based on the device fault rate, P_f, and the desired FIT rate.

The *replication with comparison* technique is a general-purpose structure and does not demand any special design

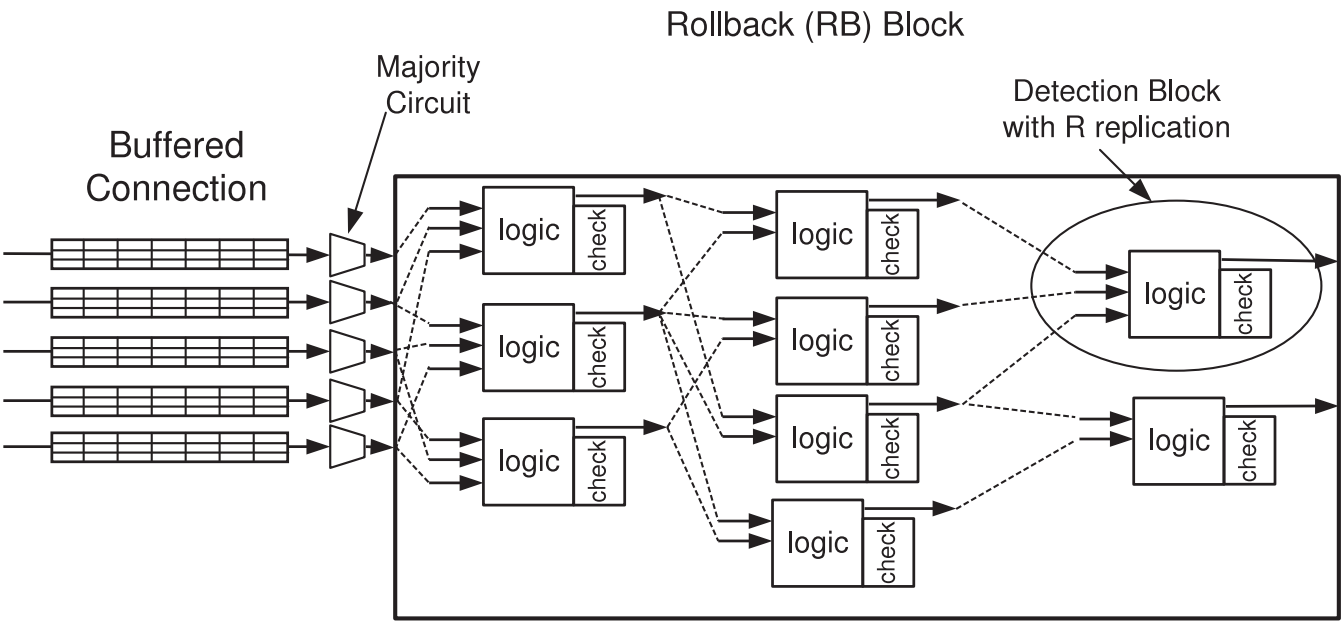

Figure 2. This figure shows an RB block consisting of detection blocks. The inputs of the RB block are buffered connections which use majority voter circuits for reliability.

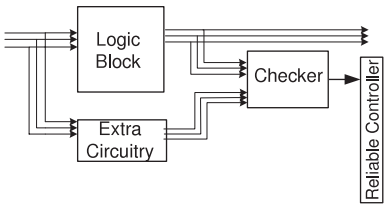

Figure 3. The checker compares the outputs of the main and extra logic and reports the error to the reliable controller.

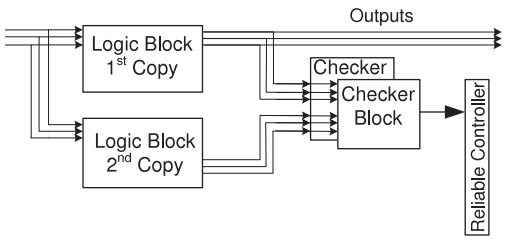

Figure 4. This figure shows a detection block. It consists of two copies of a logic block followed by two copies of a checker block.

Signals	Values		
$a_i's$	all 0	all 1	mixed
$\mathrm{OR}(a_i)$	0	1	1
$\mathrm{AND}(a_i)$	0	1	0
$\mathrm{AND}(\mathrm{OR}(a_i),\overline{\mathrm{AND}(a_i)})$	0	0	1

Figure 5. This shows the truth table of the checker block logic. The checker block reports any disagreement among the inputs, a_i's. The inputs a_i's are R copies of an output signal from a logic block. If all of the inputs hold the same value, the outputs of the AND and OR will be the same, otherwise the outputs of the AND and OR signals will be complements of each other. The last row of the table shows the error indicator function; on detecting and error, it holds the value of '1'.

specification, while design-specific alternatives (e.g. [36–40]) can be more lightweight and offer less expensive solutions. In the present paper we show that even with this basic and non-optimized detection scheme the rollback recovery will require less overhead than feed-forward fault-tolerant techniques. This makes our results conservative since more efficient, design-specific techniques can be used as applicable; further, alternate detection schemes, such as multiple parity detection [32], may also provide a path to cheaper error detection.

If the checker block is equally error prone as the logic blocks then the checker needs to be protected as well (see figure 4). Replicating the checker block and reporting an error when any of the checker block copies reports an error decreases the probability that errors in the checker will go undetected. For a limited number of errors (e.g. single error per block) when particular encodings are applicable, a self-checking checker can be used (e.g. [41, 42]) that detects errors in the input vector while experiencing limited faults (e.g. single error) inside the checker circuit. Since here we are looking at multiple errors in the checker circuit and we are already paying the replication cost for the logic block, we also use replication for the checker. As with logic replication, this also makes our results conservative; the use of more efficient checkers, when applicable, can further reduce overheads from the results we show.

A checker design which can detect any disagreement among R copies of the logic block is simple. It basically computes the AND and OR functions of the R copies of each logic block outputs. If the R signals are identical then the AND and OR functions of those signals have the same value.

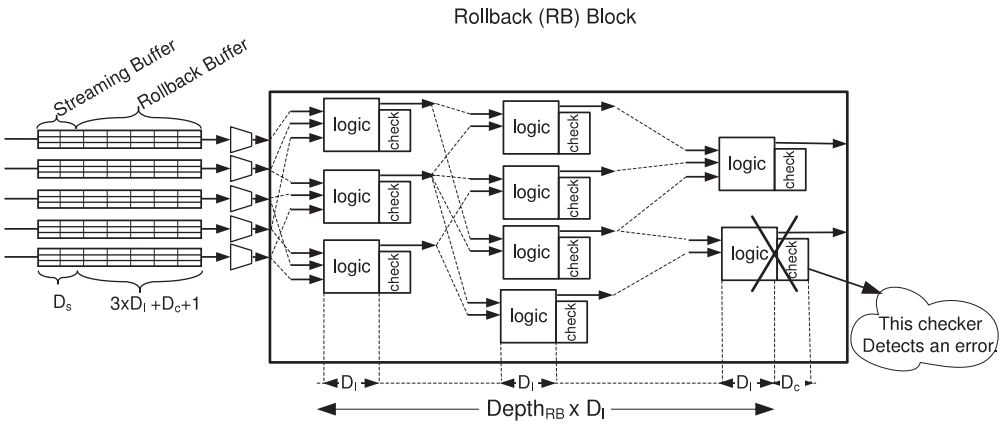

Figure 6. The input buffered connections are required to register inputs for at least $\text{Depth}_{\text{RB}} \times D_l + D_c + 1$ cycles, which is logic and checker delay plus the feedback latency.

However, if there is any disagreement between the signal values, the AND function holds 'zero' and the OR function holds 'one'. This is illustrated in figure 5 by a truth table. This implementation of the checker is minimal for the nanoPLA structure and other two-level implementations as will be shown in section 4.

A detection block is the combination of the R copies of logic blocks with the R copies of checker blocks. The structure of the detection block is shown in figure 4 for $R = 2$. In this example each detection block detects any single error and most cases of multiple errors inside the block. For any value of R, each detection block detects any $R - 1$ errors and most cases with greater numbers of errors. One important feature of this design is that the checker blocks are placed off the normal computational path, hence the latency of the checker block does not add to the latency of the normal system operation; checker latency only affects the operational latency when an error is detected.

3.2. Rollback block

When an error is detected in one of the detection blocks inside an RB block, the control circuit stops the computation of all the detection blocks inside the RB block and forces the RB block to repeat the affected process and generate the correct result. The control circuit guarantees the correctness of the rollback flow and uses the result of the checker block to switch the block operation between rollback and normal modes. The correctness of the system flow depends on the reliability of the control circuit, and therefore the control circuit must be designed with higher reliability. For example, we can implement the control circuitry with reliable, coarse-grained CMOS even when otherwise using nanoscale sub-lithographic devices for the compute block. The reliable devices take greater area but since the control circuit is a small fraction of the detection block, its area overhead is negligible compared to the area of the compute blocks.

When an error is detected, the reliable controller stops the normal operation of the circuit, resets the pointer of the input buffer to the input data associated with the last correctly retired output, and recomputes the operation from that state to recover

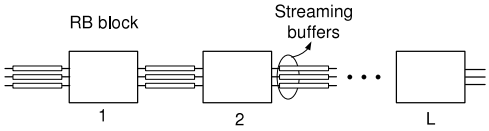

Figure 7. A simple structure model designed with buffered connections between RB blocks.

the corrupted data. How far the inputs roll back depends on the depth of the RB block and the latency of the logic blocks and checker blocks.

Figure 6 illustrates the latencies of different parts of an RB block that affect the rollback design. When an error is detected, the detection is delayed by the checker block latency (D_c cycles). Furthermore the data needed to recover the erroneous computation may have come from the RB block inputs after multiple levels of the logic block latency, e.g. figure 6 shows a case where the error is detected 3 levels deep in the block and each level has a delay of D_l cycles. So the inputs should rollback for $3 \times D_l + D_c + 1$ cycles (with one extra cycle being for the reliable controller to perform the feedback). In general the inputs of the RB block must be registered to support correct rollback operation for the following number of cycles:

$$D_R = \text{Depth}_{\text{RB}} \times D_l + D_c + 1 \tag{2}$$

where Depth_{RB} is the number of levels in the RB block (figure 6). Therefore we need a D_R-deep buffer for any of the RB block inputs. We call these D_R buffers, *rollback buffers*.

The system runs fully pipelined at high throughput until an error is detected. Then the system freezes and spends a relatively long time (i.e. $D_R = \text{Depth}_{\text{RB}} \times D_l + D_c + 1$) recovering from the error. Although this situation happens infrequently, it can have a severe impact on the system throughput. In the next section we describe how streaming buffer interconnects reduce the impact on the system throughput.

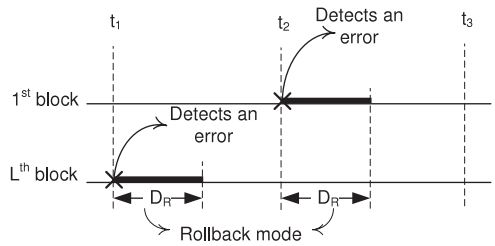

Figure 8. This is a timing diagram of the system. It shows a simple scenario where the streaming buffer can improve throughput. The Lth block detects an error at time t_1 and stays in rollback mode for D_R cycles, then the 1st block detects an error at time t_2 and switches to rollback mode.

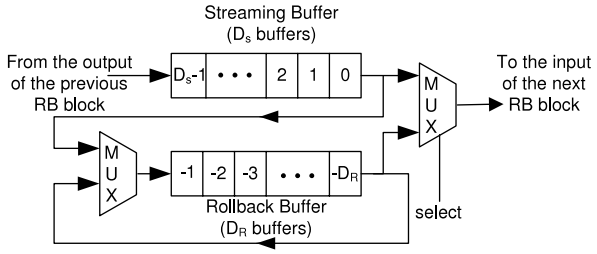

Figure 9. This figure shows a simple block diagram of a buffered connection. Each buffered connection consists of two parts: the rollback buffer and the streaming buffer. The numbers on the buffer elements represent the order of the data, '0' representing the data currently being processed in the RB block following the buffer. Each of the buffer elements in the streaming buffer or rollback buffer has a structure similar to figure 10.

3.3. Streaming buffer

When an RB block stops to rollback, the other RB blocks in the system must also stop due to the data dependences between the blocks. Consequently the system throughput drops to zero whenever any of the blocks is in rollback mode. In large systems with many RB blocks, this can potentially cause high throughput loss.

In order to avoid much of this throughput loss in large systems, we use streaming connections or *streaming buffer* between the RB blocks. The *streaming buffers* allow most of the RB blocks to continue their normal processes while some of them are in rollback mode. Note that the *streaming buffer* are extra buffers added to the required *rollback buffers* of size D_R (equation (2)). For example, if a *streaming buffer* of depth D_s is embedded at the inputs of an RB block, then the total depth of the buffer at the inputs of this block is $D_s + D_R$.

To build intuition on how the streaming buffers prevent throughput loss, we consider a simple chain structure as an example (see figure 7). This structure is also considered in [3] and [4]. It is a chain of L levels of RB blocks separated by an adequate number of buffers. Specifically let us consider a simple scenario that reveals the improvement in throughput due to the streaming buffers. Assume some errors are detected inside the Lth and the 1st RB blocks and they start the rollback process at time t_1 and t_2, respectively (figure 8). If the rollback time takes D_R cycles, in the case of no streaming buffer

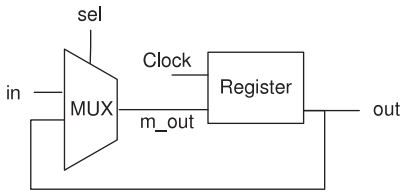

Figure 10. This figure shows a simple block diagram of a buffer element.

the system is idle for $2 \times D_R$ cycles. Therefore the system throughput during t_1 to t_3 is $(t_3 - t_1 - 2 \times D_R)/(t_3 - t_1)$.

In the presence of streaming buffers the blocks before the Lth block continue their normal process while the Lth block is in rollback mode from t_1 to $t_1 + D_R$ and the data is stored in the intermediate buffer between the $L - 1$st and the Lth blocks. Later, when the first block is in rollback mode during t_2 to $t_2 + D_R$ the Lth block continues its normal processes by consuming the saved data in the buffer between the $(L-1)$st and the Lth blocks. Therefore the total throughput loss is only D_R cycles, and the throughput during this period is $(t_3 - t_1 - D_R)/(t_3 - t_1)$. The streaming interconnects allow the blocks in the chain to run more independently and therefore, as you can see, the final throughput of $(t_1 - t_3 - D_R)/(t_1 - t_3)$ is the same as the average throughput of a single block. That is, the streaming buffers reduced the throughput loss by half in this example. Section 6 uses a simulation to estimate the best depth of the streaming buffers, D_s, to achieve acceptable throughput with reasonable area overhead.

3.3.1. Reliable buffered interconnect. Each buffered interconnect consists of two parts: streaming buffer and rollback buffer, each similar to a shift register of length D_s and D_R, respectively. Figure 9 shows how the two shift registers are connected to generate the buffer structure. In normal operation mode the data flows through the streaming buffer. One new data value is shifted into the streaming buffer from the previous RB block and one data is shifted out to the next RB block. As long as both the previous and the next RB block are in normal operation mode, the number of data elements in the streaming buffer stays the same. The number of data elements in the streaming buffer can be anything from 0 to $D_s - 1$.

If the next block detects an error and starts the rollback operations, it will stop consuming data from the streaming buffer and will start consuming the data from D_R cycles ago which is stored in the rollback buffer. During the period that the next RB block is in rollback mode, the previous RB block continues generating data and storing it in the streaming buffer until it fills up.

The streaming and reliable buffers are each composed of a chain of buffer elements shown in figure 10. Each buffer element consists of a register and a multiplexer (figure 10). The multiplexer allows either the new input or the current value into the buffer. If the buffer is in *shift* mode, the multiplexer selects the new input value, which replaces the current value. If the buffer is in *keep* mode the multiplexer selects the current value and the current value will be restored.

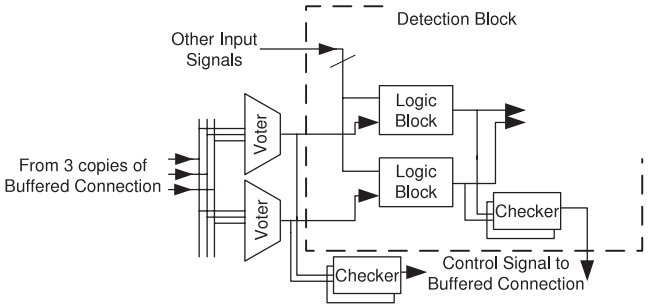

Figure 11. This figure shows how the reliable voters are structured and connected to separate replicas of logic blocks in each detection block.

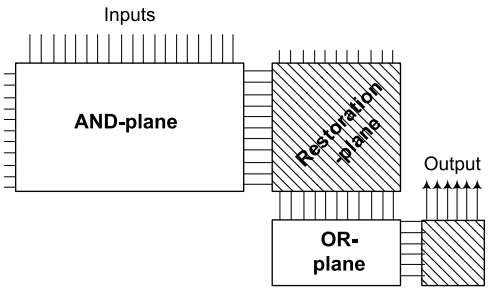

Figure 12. This shows a functional view of a nanoPLA block [6].

The data coming out of the buffered connections into the RB block must be correct. To guarantee the correctness of the buffered connection data, an error correcting technique is embedded in the buffers.

For simplicity and consistency with the error detection technique in the logic blocks we use the majority voting scheme for error correction in the reliable buffer. In this scheme multiple copies of the data are stored and a voter circuit following the multiple copies determines the majority among these copies. This scheme needs large data redundancy (i.e. minimum of 3) but the encoder (replicator) and the decoder (voter circuit) are relatively cheap when the replication factor is small.

We call the replication factor for each buffer element R_{buf}. The minimum R_{buf} for majority voting is 3 and it grows for high fault rates. The voter circuit receives all the R_{buf} copies of the buffer element. It computes the majority of the R_{buf} input signals. This is the value of at least $(R_{\text{buf}}/2 + 1)$ of the inputs.

If there were a single voter circuit for every R_{buf} copies of the buffered data, the voter circuit would be a single point of failure and the reliability bottleneck; the reliability improvement achieved by multiple copies of the buffer element will be wasted. To prevent this effect, the computation of the voter circuitry must also be protected. Therefore, similar to the logic blocks the voter circuit is replicated into R copies and the correctness of the results is verified by checker blocks following them (figure 11). When a checker block identifies a disagreement among the voter results, the recovery process is similar to the case when an error is detected in a logic block; that is, the process of the following RB block is stopped and the voter circuits repeat the operation to identify the correct value of the majority of the incoming signals from the buffered connections.

3.4. Block size

Key parameters in rollback system design are the detection block size and RB block size. The detection block affects the likelihood of detecting transient faults and, hence, determines the reliability of the system. The RB block size controls the latency of rollback and the rate at which rollback occurs and, hence, is largely responsible for determining the throughput of the system. By treating the detection and RB block sizes independently, we can separately engineer the system for reliability and performance. Both block sizes affect the overhead in the system.

As we will see in section 5 the reliability of a detection block for a fixed device fault rate depends on the replication factor and the block size. Larger replication factors and smaller block sizes increase the reliability of the detection block. Therefore, for a fixed reliability target and device fault rate, we have to limit the detection block size to keep the required replication factor small. Nevertheless interconnect locality, fixed block overheads and reliable control circuitry make the smallest block sizes (e.g. single product terms or even product terms with only two inputs) inefficient [43]. Therefore there is a practical lower bound on efficient block sizes. The area minimizing block sizes for various nanoPLA designs is shown in [43]. These efficient designs have fine-grained block size (i.e. logic-level). Here we try to design the rollback system where the size of the detection blocks is close to this efficient size.

The RB block size affects the throughput and area overhead of the rollback system. The impacts of the RB block size are summarized in the following categories:

(i) In rollback mode, the operation of the block will be recomputed. The main part of the rollback latency is the latency of the main block, which was shown in equation (2). Small block size, or more specifically small block depth, Depth_{RB}, helps keep the rollback latency short and, in turn, keeps D_{R} small.

(ii) The larger the block is, the higher the probability of transient fault occurrence in the block, and therefore the higher rollback frequency. If the device failure probability is P_{f} and the block has N devices, the block fails with the probability below if we ignore fault masking:

$$P_{\text{RB}} = 1 - (1 - P_{\text{f}})^N. \tag{3}$$

When $N \times P_{\text{f}} \ll 1$, the failure probability is approximately $N \times P_{\text{f}}$ and we see that rollback frequency grows linearly with the size of the block.

(iii) The RB block size also affects the area overhead, but in different directions. Larger block size results in smaller area overhead by reducing the number of buffered connections. Large blocks tend to enclose connections between the detection blocks inside it, thus reducing the number of inter-RB-block connections which are implemented in buffered connections.

As you can see, the first two effects above favor small RB block size to achieve high system performance, while the last one favors large RB block size to reduce area overhead; this suggests the RB block size selection provides a tradeoff between area and time. When fault rates are low, we can employ large RB block sizes to minimize area overhead, but as fault rates increase, the RB block sizes must decrease to maintain performance, at the cost of additional area overhead. Section 6 quantifies this tradeoff.

Note that the optimum size of the RB block is much larger than the detection block size. This is the main motivation for designing a fine-grained rollback system in two hierarchical levels with two different block sizes (see table 2). We can have larger RB blocks which amortize the overhead of streaming inputs without decreasing reliability or increasing error detection overhead.

4. NanoPLA implementation

In this section, the implementation of the fine-grained streaming rollback design will be demonstrated on a nanoPLA substrate. Before going into our rollback design implementation on the nanoPLA architecture, a brief overview of this architecture will be shown here; a more detailed description of this architecture is available in [18].

The nanoPLA architecture is similar to the conventional PLA (programmable logic array). Each nanoPLA block realizes a two-level logic circuit (i.e. sum of products). A functional view of a nanoPLA block is shown in figure 12. The inputs enter the AND-plane. This plane generates the product terms, Pterms. The Pterms pass through a first restoration plane to restore their voltage level. The restored Pterms enter the OR-plane to generate the outputs of the two-level logic, OR-terms. These OR-terms then pass through the second restoration plane and make the final outputs. In each nanoPLA block, the Pterms and the OR-terms are implemented in wired-OR logic using nanowires, and the controllable junctions are diode-like switches placed at the intersection of two nanowires [19]. The restoration elements are made of modulation doping along a nanowire [22, 44]. NanoPLA blocks can be interconnected using nanowires. The input nanowires enter the AND-plane vertically and the output nanowires exit the restoration plane following the OR-plane (figure 12). The same nanowires in the AND- and OR-planes are used to route the signals between the blocks; as a result, there is no difference between routing and computational resources.

As explained above the nanowires operate in pairs; the nanowires in the logic plane generate the wired-OR logic and the nanowires in the following restoration plane invert their value and restore their voltage level. We consider each pair of nanowires and the corresponding input diode switches and the gate-controlled junction in between nanowires as a unified element. We define the fault rate, P_f, the probability that this unified element is erroneous. We also measure the area of our system based on the number of nanowire pairs.

The nanoPLA block described in [18] serves as a template and framework for containing nanowire pairs and includes the infrastructure for testing, programing, powering and clocking

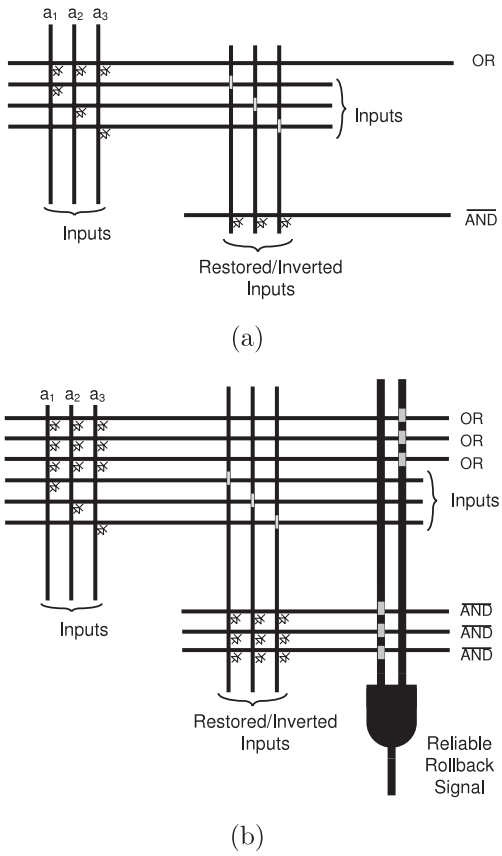

Figure 13. (a) This figure shows the checker block implemented with nanowires. As you can see from this example the nanoPLA checker block takes $R + 2$ ($R + 2 = 5$ in this example) nanowire pairs. (b) This figure shows how the R copies of the checkers are integrated with the thin slice of reliable lithography-scale circuitry.

the nanowire logic. Detailed designs in [18] compute the area for nanoPLA blocks and identify the nanoPLA organizations which require the least total area. Typical nanoPLA blocks hold around 100 nanowire pairs and are roughly 10 μm wide and 5 μm tall. In practice, we would use a single, homogeneous nanoPLA block organization in a chip-scale array and distribute the Pterms of a design across the nanoPLA blocks in the array, where each Pterm is mapped to a pair of nanowires. While logic clustering efficiency is important and can fluctuate within a design, the number of nanoPLA blocks required for a design is roughly proportional to the number of nanowire pairs; consequently, for this work we count nanowire pairs to estimate the relative areas of various designs.

4.1. Detection and rollback block

The detection block developed in the previous section is implemented on the nanoPLA substrate. Multiple logic blocks may be implemented by each nanoPLA block; each logic block is replicated R times and followed by the checker blocks which are also implemented in nanoPLA blocks. The checker function consists of an R-input AND function and an R-input OR function and can easily be implemented in two-level logic as described in section 3.1. Figure 13 shows the checker design implemented in a nanoPLA block with $R = 3$. The nanoPLA

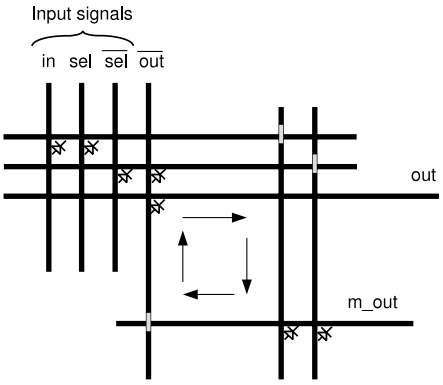

Figure 14. This figure shows a shift register element implemented with nanowires. The schematic view of this design is shown in figure 10. The *m*_out signal is an intermediate signal (output of the MUX), which is routed into the input plane to generate the final output signal. This implementation needs four pairs of nanowires.

checker block needs one Pterm to implement the R-input OR function and R Pterms and one OR-term to implement the R-input AND function (figure 13). Overall a checker circuit needs R Pterms and 2 OR-terms to check the agreement between R signals, which in total takes $R+2$ pairs of nanowires.

Since the checker size is relatively small the R copies may be integrated into one nanoPLA block. As shown in figure 13, the R copies of the checker takes, $R \times (R+2)$ nanowire pairs.

The final outputs of the checker block connect to reliable control circuitry through a wired-OR (figure 13) to generate the final reliable feedback control signal. That is, we want to signal a rollback when any of the checker outputs signals an error; the nanoscale checker outputs are wired via diode connections to a reliable, lithographic scale wire so that it is pulled high when any of the checker outputs is high. Strictly speaking, the efficient implementation shown in figure 13(b) implements $(\overline{and_0} + \overline{and_1} + \overline{and_2}) \cdot (or_0 + or_1 + or_2)$ rather than $\overline{and_0} \cdot or_0 + \overline{and_1} \cdot or_1 + \overline{and_2} \cdot or_2$, where and_i's are the AND's and or_j are the OR's; the extra cross terms should also always be zero in a fault-free case, so these additions do not cause any false rollbacks.

The detection blocks, including logic blocks and checker blocks, are clustered to form an RB block. The interconnect signals among the detection blocks inside an RB block are routed in the bundle of R nanowires. The interconnect signals are implemented on the nanoPLA planes. The details of how interconnect routing can be implemented on nanoPLA planes are provided in [18].

4.2. Buffer connection

The buffered connection, as described in section 3.3.1, is a chain of buffer elements, each consisting of a multiplexer and a register. Figure 14 shows how this can be implemented on a nanoPLA substrate. The details of the buffered connection implemented on the nanoPLA can be found in [18]. This design takes 4 pairs of nanowires per cell and multiple buffer elements can be implemented in one nanoPLA plane.

The voter circuit following a buffered connection is an OR function of all the possible $(R_{\text{buf}}/2+1)$-input AND gates from

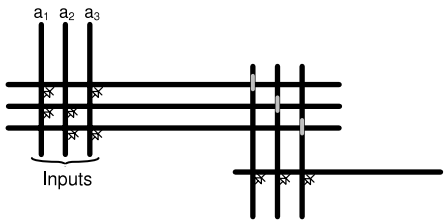

Figure 15. This figure shows the voter circuit for redundancy factor $R_{\text{buf}} = 3$, designed with nanoPLA.

R_{buf} signals. Therefore the number of AND gates in the voter circuit is

$$A_{\text{maj}}(R_{\text{buf}}) = \begin{pmatrix} R_{\text{buf}} \\ (R_{\text{buf}}/2+1) \end{pmatrix}. \quad (4)$$

When R_{buf} is small, the above number is not very large. For large values of R_{buf}, there are alternate options that can provide more compact implementations (as small as $O(R_{\text{buf}})$) at the expense of greater checker latency, D_c. Figure 15 shows the voter circuit for $R_{\text{buf}} = 3$. It has 3 AND gates (Pterms) followed by an OR-term.

Using the above design, the number of nanowire pairs required for a buffered connection of depth $D_R + D_s$ including R copies of the voter circuit is

$$A_{\text{buffer}} + A_{\text{vote}} = \text{Size}_{\text{buf}} \times (D_R + D_s) \times R_{\text{buf}} + R \times \begin{pmatrix} R_{\text{buf}} \\ (R_{\text{buf}}/2+1) \end{pmatrix}$$

where Size_{buf} is the number of nanowire pairs in one buffer element which equals 4.

4.3. Rollback and stream control

The rollback control for a rollback block is implemented at the lithographic scale with lithographic inputs and outputs to the nanoPLA blocks. The total resources required for this control are:

(i) Additional lithographic signals in most nanoPLA blocks either for:
  (a) buffers—stall and rollback inputs (controlling the multiplexers in figures 9 and 10);
  (b) checkers—the pair of lithographic wires to support the reliable rollback signal shown in figure 13.

(ii) Fanin of rollback and stall signals to rollback control block and fanout of stall and rollback signals from controller to stream buffers.

(iii) Rollback control logic.

Overall, this control logic has modest impact compared to the area required for replication, checkers and reliable stream buffers.

(i) Recall that the nanoPLA blocks are roughly 10 μm $\times$ 5 μm; as such, adding a pair of lithographic wires (e.g. 105 nm pitch each for the 45 nm lithographic node) increases the area of a nanoPLA block by $\approx$2%.

(ii) A tree has fewer internal edges than external edges, so the fanin and fanout trees add less than three lithographic fanin/fanout nodes per nanoPLA block (error detection, rollback and stall). Further, a modest rollback block (see table 2) composed of 3200 = 40 × 80 nanoPLA blocks will be 400 μm × 400 μm, meaning the lithographic wire runs are relatively short.

(iii) This control logic is amortized across all the nanoPLA blocks (see table 2). With hundreds to thousands of control blocks, the area of even a thousand lithographic scale gates to implement this control is small compared to the total area of the nanoPLA blocks.

5. Reliability and area analysis

In this section we analyze the area and reliability of our fault-tolerant approach. The main goal in this section is to determine how large the replication factor must be to achieve a desired FIT rate. To do so, this section is organized as follows: we first compute the *undetected error probability* of the system using a bottom-up approach, i.e. we compute the undetected error probability of the building blocks of the system from the base-level *detection block*, to the *RB block*, to the complete system. Once we have the undetected error probability of the system and know the system frequency, we can compute the expected number of undetected errors in one billion operational hours, which is the FIT rate of the system.

5.1. Error probability of a detection block

To compute the undetected error probability of a detection block, we first have to compute the error probability of its building blocks: *logic blocks* and *checker blocks*.

Here we consider each logic block as the *logic cone* of each output signal. The *logic cone* of an output signal is the set of all the logic elements required to generate the output signal and therefore is the only part influencing the output signal.

With a conservative estimate, an OR-term (an output signal of a logic block in the nanoPLA architecture) has an erroneous result if any element inside the block is erroneous. It is conservative since it does not consider the effects of any kind of error masking, e.g. *logic masking*, *electrical masking* or *latching-window masking* [45]. *Logic masking* is when the error might not propagate to the output because a gate on the path is not being sensitized to facilitate the propagation. *Electrical masking* is when an error is attenuated passing through multiple gates on the path to the output. Finally *latching-window masking* is when the fault effect reaches the output but the latch is not open to store the erroneous value.

Using this conservative assumption, any fault in the logic block will result in an error in the OR-term signal. Therefore the probability that an OR-term has an erroneous value is

$$P_{\text{or\_err}} = 1 - (1 - P_{\text{f}})^{N_{\text{logic}}} \tag{5}$$

where N_{logic} is the size of the logic cone of the OR-term. With a similar calculation the error probability of a checker block is

$$P_{\text{cb\_err}} = 1 - (1 - P_{\text{f}})^{R+2} \tag{6}$$

where $R + 2$ is the size of the checker block as shown in section 4.1.

Now that we know the error probability of building blocks of a detection block, we can compute the probability of an undetected error in a detection block. In a detection block with R copies of a logic block and R copies of a checker block, an erroneous OR-term is undetected under two scenarios: first, when all the R copies of the OR-term are erroneous and all the checker blocks are correct, in this case no disagreement among the OR-term copies can be detected; second, when at least one of the OR-term copies are erroneous but all the R checker copies are erroneous and fail to detect the error. These two cases generate the undetected error probability of a detection block as below:

$$P_{\text{det\_block\_und\_err}} = (P_{\text{or\_err}})^R \times {P_{\text{cb\_crr}}}^R + (1 - (P_{\text{or\_crr}})^R)(P_{\text{cb\_err}})^R. \tag{7}$$

Note that $P_{\text{or\_crr}}$ and $P_{\text{cb\_crr}}$ are the probability that an OR-term signal or a checker block is correct; these are the complement of $P_{\text{or\_err}}$ and $P_{\text{cb\_err}}$, respectively, which are computed in equations (5) and (6).

Remember that the reliability of the voter circuitry following each buffered connection at the input of an RB block is provided by replication of the checker circuitry. Therefore the voter circuitry generates an undetected error in the same scenario as a logic block in a detection block does: (1) when all the R copies of the voter circuitry are erroneous, which results in identical erroneous output signals, and all the checker copies are correct; (2) when at least one of the R copies of the voter signal is incorrect but all the checker copies fail to detect the erroneous voter circuit copy. This probability is similar to equation (7):

$$P_{\text{vote\_block\_und\_err}} = (P_{\text{vote\_}err})^R \times {P_{\text{cb\_crr}}}^R + (1 - (P_{\text{vote\_crr}})^R) \times (P_{\text{cb\_err}})^R. \tag{8}$$

$P_{\text{vote\_crr}}$ and P_{vote_err} are the probabilities that a voter circuit is error-free or erroneous, respectively, which is essentially the same as a logic block's with $N_{\text{logic}} = \binom{R_{\text{buf}}}{(R_{\text{buf}}/2+1)}$ nanowire pairs.

5.2. Undetected error probability of an RB block

Each RB block includes a number of detection blocks. It also includes a number of voter blocks following any incoming buffered connection. An RB block has an undetected error in it if any of its detection blocks or the voter blocks has an undetected error. Therefore the undetected error probability of an RB block with B detection blocks and I inputs is

$$P_{\text{rb\_block\_und\_err}} = \left(1 - (1 - P_{\text{det\_block\_und\_err}})^B\right) \bigcup \left(1 - (1 - P_{\text{vote\_block\_und\_err}})^I\right). \tag{9}$$

Note $\bigcup$ is used here to denote a probability union calculation, where we avoid counting the overlap probability twice; that is:

$$A \bigcup B \equiv A + B - A \cdot B. \tag{10}$$

5.3. Buffered connection reliability

The error probability of a buffer element depends on the number of consecutive cycles that a buffer element holds a single logic value in the system and therefore it is susceptible to errors. In order to have a realistic estimate on the number of consecutive cycles that a buffer element holds a single value, we simulate the performance of the system. This simulation is explained in section 6 for the same chain structure introduced in section 3. The error probability that a buffer element has an erroneous value in a single cycle is

$$P_{\text{buf\_elem\_err\_per\_cycle}} = 1 - (1 - P_{\text{f}})^{\text{Size}_{\text{buf}}} \tag{11}$$

where Size_{buf} is the number of devices in one buffer element. Once we have the maximum number of consecutive cycles that a buffer element holds a single value, we can compute the error probability of a buffer element as below, where c is the number of those cycles:

$$P_{\text{buf\_elem\_err}} = 1 - (1 - P_{\text{buf\_elem\_err\_per\_cycle}})^{c}. \tag{12}$$

A protected buffer element with replication (R_{buf}) has an undetected error when the number of erroneous replicas is more than half of the replication factor (R_{buf}), and therefore the majority computes the wrong value. This probability is written below:

$$P_{\text{buf\_und\_err}} = \sum_{i=\lceil R_{\text{buf}}/2 \rceil}^{R_{\text{buf}}} \binom{R_{\text{buf}}}{\text{i}} P_{\text{buf\_elem\_err}}{}^{i} \times (1 - P_{\text{buf\_elem\_err}})^{R_{\text{buf}}-i}. \tag{13}$$

5.4. Undetected error probability of the complete system

The undetected error probability of the system will be computed similarly to the undetected error probability of an RB block. There is an undetected error in the system if there is an undetected error in any of the RB blocks of the system or any of the buffered connections of the system. An undetected error in an RB block results from an undetected error in its constituent detection blocks, and an undetected error in a buffered connection results from an undetected error in any of its constituent buffer elements. Therefore we can conclude that any undetected error in the system results from either an undetected error in any of the detection blocks or the buffer elements of the system. In a system with a total of S_{D} detection blocks and S_{B} buffer elements, the probability that the system has at least one undetected error is

$$P_{\text{sys\_und\_err}} = \left(1 - (1 - P_{\text{det\_block\_und\_err}})^{S_{\text{D}}}\right) \bigcup \left(1 - (1 - P_{\text{buf\_und\_err}})^{S_{\text{B}}}\right). \tag{14}$$

Equations (5)–(14) develop the undetected error probability in the whole system. Once we have the undetected error probability of the whole computation and having the system frequency, we can compute the FIT rate of the system, which is the number of undetected errors in 10^9 h of system operation:

$$\text{FIT} = P_{\text{sys\_und\_err}} \times 10^9 \times \text{System frequency}. \tag{15}$$

Later in this section, using the above analysis, we show the required replication factor of R for a sample system specification. The complete area overhead including the buffered connections will come in the following section, at section 6.

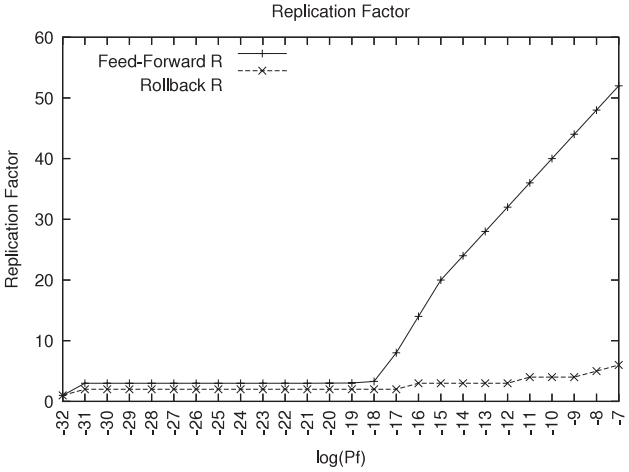

Figure 16. This graph compares the replication factor of rollback and feed-forward recovery. The feed-forward recovery data is from the majority multiplexing shown in [4]. In their analysis the system reliability goal is 90%. We recomputed their results for a system reliability of ($P_{\text{sys\_und\_err}} = 1\text{–}10^{-20}$), which is equivalent to a FIT of 360 used for our system specifications.

5.5. Redundancy analysis

Using the above analysis, we show the required replication to achieve the desired FIT rate for a sample system. In this section we focus on the logic replication factor R and compare this value with a feed-forward fault tolerant approach. The detailed complete area overhead analysis including the buffered interconnect will be shown in the next section.

In order to use equations (5)–(14), we have to specify the following system parameters:

- N_{logic}, logic block (logic cone) size: The logic block size depends on the design substrate. For the nanoPLA architecture model, we identify the efficient logic block sizes for permanent defect tolerance in [46]. In [46] we bound the mapping redundancy for defects by limiting the fanin size of each OR-term. From the experiments in [43], we see that a logic block size of $N_{\text{logic}} = 16$ achieves compact systems close to the minimum size. Here we keep the same $N_{\text{logic}} = 16$ in our analysis since it is small enough to minimize the replication factor, R, as explained in section 3.4.
- S_{D}, the system size: The value of S_{D}, the number of detection blocks in the system, can be computed from the total number of devices in the system, N_{t}, divided by the size of a detection block. The size of a detection block is $R \times (N_{\text{logic}} + (R + 2))$, consisting of R logic blocks and R checker blocks. Estimating the number of devices in the system built on the nanoPLA substrate, excluding the buffered connections, around $N_{\text{t}} = 10^{12}$, the number of detection blocks in the system would be:

$$S_{\text{D}} = N_{\text{t}}/(R \times (N_{\text{logic}} + (R + 2))) = 10^{12}/(R \times (16 + (R + 2))).$$

The size of S_{B}, the number of buffer elements in the system, is determined through the simulation described in section 6.

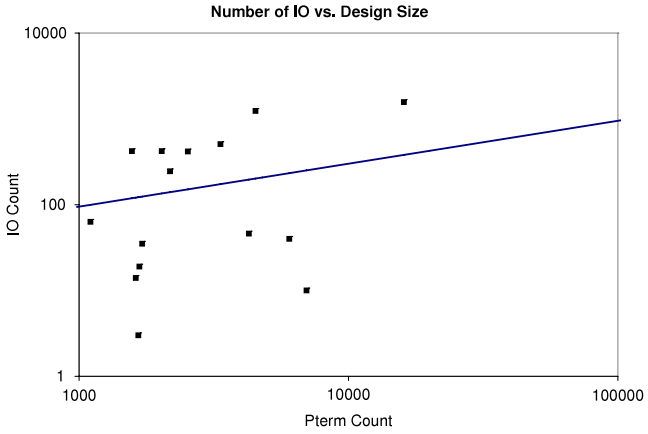

Figure 17. This graph shows the number of primary inputs and outputs (IO) versus the number of Pterms in a design. The data is from the toronto20 benchmark set implemented on the nanoPLA substrate with logic block size of 16. The curve shows the exponential function fitted to the data points, which is $\mathrm{IO} = 3.2 \times (\mathrm{Pterms})^{(0.51)}$.

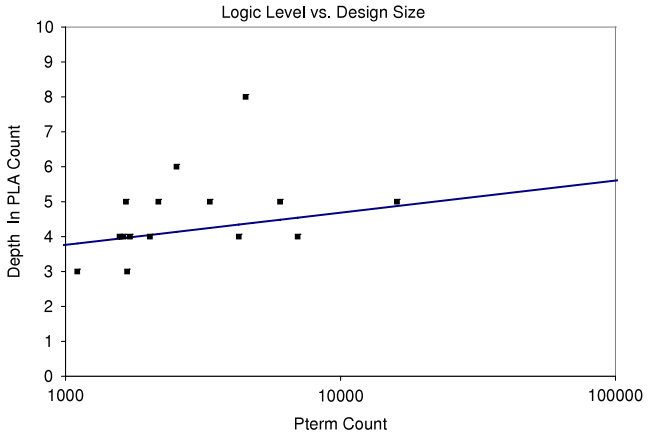

Figure 18. This graph shows the depth of the design in the number of nanoPLA planes versus the number of Pterms in the design. The data is from the toronto20 benchmark set implemented on the nanoPLA substrate with logic block size of 16. The curve shows the logarithmic function fitted to the data points. This function is $\mathrm{Depth} = 0.92 \log_{10} (\mathrm{Pterms})$.

- The system frequency: The system runs at 10 GHz frequency, which is a reasonable expectation for future system design.
- Desired FIT rate: The desired FIT rate in this example is 360. With the above system frequency of 10 GHz the undetected error probability for the system will be $P_{\mathrm{sys\_und\_err}} = 10^{-20}$.
- P_{f}, device failure rate: The device failure rate ranges from 10^{-32} to 10^{-7} similar to previous studies [4, 7].

We compare our rollback recovery results with feed-forward recovery results of [4] (reviewed briefly in section 2.4). In [4] the analysis was done for a system reliability rate of 90%. Here we perform the calculation in [4] with the new $P_{\mathrm{sys\_und\_err}} = 10^{-20}$ (for FIT = 360 and system frequency of 10 GHz), which is much lower than the 10% target used in [4].

Figure 16 plots the value of R for different values of P_{f}. These curves compare the replication factor of rollback recovery and feed-forward recovery. For fault rates smaller than 10^{-32} the system with no protection satisfies the system reliability goal of $(1–10^{-20})$. For higher fault rates just above 10^{-32} (left side of the graph) the rollback recovery has a replication factor of two (the minimum replication factor for error detection) and the feed-forward recovery has a replication factor of three (the minimum replication factor for the majority multiplexing feed-forward recovery technique as described in section 2.4). As the fault rate increases the gap between the rollback and the feed-forward technique increases. The gap starts to grow dramatically for P_{f} larger than 10^{-18}. The feed-forward replication factor grows to almost an order of magnitude greater than rollback recovery for $P_{\mathrm{f}} \geqslant 10^{-9}$.

In this section we analyzed the replication factor of the rollback technique and demonstrated that the rollback recovery technique requires about one order of magnitude lower replication factor than the feed-forward recovery technique. In the next section we see how the complete area including the checker and the buffered connections compare against the feed-forward recovery technique. We also estimate the system throughput and see how rollback impacts the system performance.

6. Simulation and comparison

In this section, we simulate our proposed reliable technique in the presence of random transient faults with various fault rates. We measure the system throughput and demonstrate the complete area overhead including the checker blocks and the buffered connections area.

6.1. Modeling parameters

There are two variable parameters in our system specification that need to be specified to achieve the desired area–time tradeoff: the RB block size and the streaming buffer depth. The RB block size, as explained in section 3.4, has two different effects on the system: First, larger RB block sizes enclose more interconnects inside them and therefore reduce the total number of buffered connections in the system. As a result larger RB blocks allow compact system implementation. The second phenomenon has the opposite effect; larger RB blocks tend to have more logic levels in the block, which increases the rollback latency, and a higher frequency of rollbacks. Therefore using smaller RB blocks results in higher system performance. In our simulation we will find the best RB block size which balances these effects to minimize the area overhead and maximize the system throughput.

In order to meaningfully estimate the number of interconnects and the number of logic levels in an RB block, we tune our estimation with the *toronto20* benchmark set [47]. We map the designs in this benchmark set to nanoPLAs using a logic block size, N_{logic}, of 16. Figures 17 and 18 show the results of this mapping. Figure 17 shows the number of primary inputs and outputs of a design as a function of the design size in Pterms. In this figure, each data point

Table 1. This table shows the depth, D_s, and the replication factor of buffered connections, R_{buf}.

$\log(P_f)$	$\leqslant -16$	-15	-14	-13	-12	-11	-10	-9	-8	-7
D_s	1	1	1	1	1	1	2	2	3	3
R_{buf}	3	5	5	5	5	5	7	7	9	9

Table 2. This table shows the number of detection blocks in an RB block.

$\log(P_f)$	RB block size	P_{detect}
$\leqslant -11$	10 000	$\leqslant 9.3 \times 10^{-6}$
-10	3 500	3.2×10^{-5}
-9	3 000	2.7×10^{-4}
-8	2 500	3.1×10^{-3}
-7	300	4.9×10^{-3}

represents a design from the benchmark, and the trend shown is a fitted Rent's rule [48] curve (i.e. $IO = c \cdot (N_{blocks})^p$) to the data points. Similarly, figure 18 shows the logic depth of a design as a function of the design size. The data points represent the designs from the benchmark and are fitted to a logarithmic curve. In our simulation, we use the fitted curves from figures17 and 18 to estimate the number of buffered connections at the boundary of an RB block or the number of logic levels in an RB block, respectively.

6.2. Streaming buffer simulation

We simulate the throughput of the system on the chain structure introduced in section 3.3. The building blocks of the chain are RB blocks and the length of the chain is 100 blocks. This is the same structure that was used in [4] to estimate redundancy factors required in the feed-forward approach.

The rest of the system parameters are the same as the previous section: $N_t = 10^{12}$, the system frequency is 10 GHz, and the FIT rate is 360.

During the simulation, random faults are injected into the system with probability of P_f. For each P_f we use the simulator to examine a range of RB block sizes and pick the best RB block size. For each RB block size we compute the area overhead and simulate the system throughput; this operation starts with the streaming buffer depth $D_s = 1$ and, if the throughput is not high enough, increments D_s by one for each trial until the desired throughput is achieved. Here we set our throughput threshold at 98% for $\leqslant 10^{-9}$, and 90% for $> 10^{-9}$ (i.e. we add buffers until the throughput is at least 98% (or 90% for $> 10^{-9}$) of the throughput of the fault-free case). Table 2 shows the RB block sizes which achieve the minimum area overhead while keeping the throughput above 98% (or 90% for $> 10^{-9}$). Table 1 shows the required streaming buffer depth to achieve the throughput target.

The RB block size and transient error rates determine the probability that each RB block detects an error and rolls back and, consequently, determines the throughput sustainable by the RB block. Table 2 shows the probability that an RB block detects an error (P_{detect}). For each P_f the RB block size is made small enough to keep P_{detect} low while not increasing the area overhead impractically large. We observed that, for low

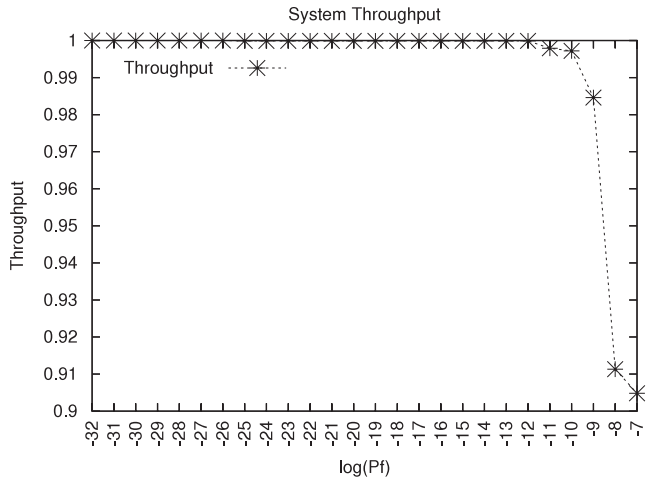

Figure 19. This graph shows the system throughput as the function of failure rate, P_f.

P_{detect}, small streaming buffer depth is required (e.g. $D_s = 1$) while larger P_{detect} demands larger streaming buffer depth. The system needs the minimum of $D_s = 1$ to achieve high system throughput even for smaller fault rates. With no buffering, a single rollback stalls all the logic on the chip; however, the elasticity provided by even the minimum size D_s limits the impacted number of RB blocks. For example, let D_s be 1 and the rollback latency (D_R) be 4 (which is the minimum rollback latency). Then if the ith RB block detects an error and stops to rollback at time t, the rollback wave expands to the $i-4$th RB block over the period of four cycles, such that the $i-1$st block runs for one more cycle after cycle t, filling up the single streaming buffer following that block and stopping at cycle $t+1$. The $i-2$nd block runs for another cycle, filling up the single streaming buffer following this block and stopping at cycle $t-2$. This continues until the $i-4$th block stops at $t+4$, after filling up its following streaming buffer. The ith block had zero throughput from cycle t to $t+4$: however, four data elements are stored in the four streaming buffers distributing over four stages. So if any block preceding the $i-4$th stage detects an error and stops to rollback in the future, the four data elements will be consumed by the following blocks, preventing these downstream blocks from sitting idle for another D_R cycles, the same effect that was shown earlier in section 3.3. Consequently, this minimum buffering guarantees that RB blocks further away in the chain are not impacted by this failure; if we see only one rollback occurring at a time, only the few RB blocks immediately adjacent to the affected RB block stall, while the majority of RB blocks continue their operation.

We observed the following interesting effect of the streaming buffer depth and the rollback block size on the

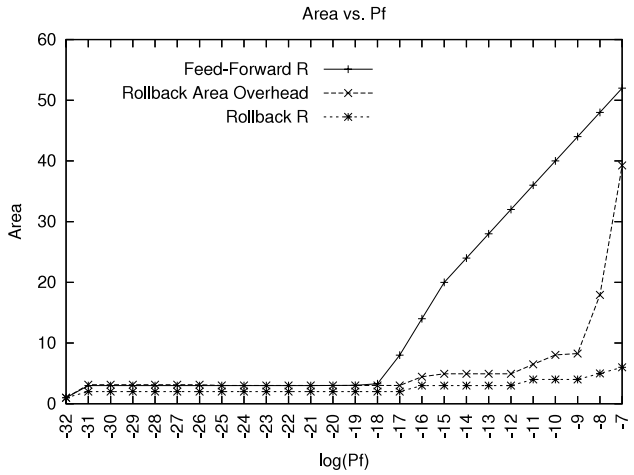

Figure 20. The solid curve with '+' markers shows the replication factor of the feed-forward technique from [4] with a higher reliability goal of FIT = 360. The curve with '∗' markers is the replication factor of the rollback recovery. The third curve with '×' markers shows the total area of the rollback recovery technique.

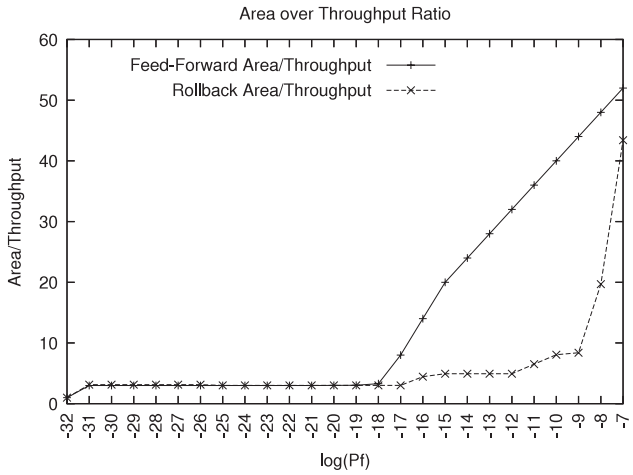

Figure 21. This graph plots the area/throughput for the rollback and feed-forward recovery techniques.

Table 3. In A_{logic} the value of B is the number of logic blocks in RB blocks. The value of N_{logic} is 16 nanowires. In A_{buffer}, IO is the number of buffered connections of an RB block, which is estimated by the curve in figure 17. For nanoPLA detection block $D_L = 1$ and $D_C = 2$. The streaming buffer depth, D_s, is defined by the throughput simulation and table 1 shows the selected values of D_s for different fault rate values, generated by our simulation.

One RB block area in the number of nanowire pairs	
A_{logic}	$R \times B \times N_{\text{logic}}$
A_{checker}	$(R-1) \times (R+2) \times B$
A_{buffer}	$\text{IO} \times R_{\text{buf}} \times \text{Size}_{\text{buf}} \times (D_s + \text{Depth}_{\text{RB}} \times D_L + D_C + 1)$
A_{voter}	$\text{IO} \times \left(\frac{R_{\text{buf}}}{R_{\text{buf}}/2+1} \right) \times R_{\text{buf}}$

Table 4. This table shows the distribution of the area over different parts of an RB block.

P_f range	A_{logic}	A_{check}	A_{buffer}	A_{maj}
$10^{(-29)}$–$10^{(-17)}$	66.56	10.40	22.51	0.52
$10^{(-16)}$–$10^{(-11)}$	60.85	15.21	22.87	1.06
$10^{(-10)}$	49.75	16.32	30.07	3.86
$10^{(-9)}$	48.40	15.88	31.66	4.06
$10^{(-8)}$	27.88	11.15	53.61	7.36
$10^{(-7)}$	15.28	7.16	67.86	9.69

system throughput: the simulation shows that the impact of RB block size on the throughput is stronger than the depth of the streaming buffers. This means that, in a nominal design, reducing RB block size yields a larger throughput improvement than increasing the depth of the streaming buffers between the RB blocks. Therefore to achieve high throughput and keep area overhead low, it is more beneficial to minimize the RB block size and use the minimum required streaming buffer depth. Note that the RB block size reduces to 300 detection blocks, or 188 000 Pterms, by $P_f = 10^{-7}$; these results show how the strong dependence of RB block size on device fault rate drives us to fine-grained rollback blocks for designs at these fault rates. We also note that, even at this high transient fault rate and relatively high rollback overhead, the RB block size does not reduce to a single detection block, underscoring the value of keeping the detection block size separate from the RB block size (section 3.4).

6.3. Area and throughput simulation results

The areas determined from the simulation are plotted in figure 20. Figure 20 shows the replication factor, R, and the total area overhead of the rollback recovery technique. The figure also plots the replication factor of the feed-forward technique for comparison. The replication factors are computed as explained in section 5.5. The total rollback area overhead curve includes the complete area of the RB blocks and the buffered connections.

Figure 19 plots the throughput of the system. As you can see for $P_f \leqslant 10^{-9}$ the impact on the throughput is almost negligible and for higher fault rate the drop in throughput is less than 10%. This minimal impact on the throughput is achieved while reducing the area required by a factor of six compared to the feed-forward recovery technique.

In order to understand the area curve in figure 20, it is helpful to understand how the system area is distributed over different parts of the system. Table 3 summarizes the equations used to compute the area of each component in an RB block; area is calculated in terms of nanowire pairs. Table 4 shows how the area of the system is distributed over different parts of the system for different fault rates, P_f. As you can see the logic and checker area is the dominant portion of the total system area for moderate fault rates ($P_f < 10^{-9}$). The buffered connection area (A_{buffer}) plus the voter area (A_{voter}) increase as the fault rate P_f increases. Achieving high throughput with high fault rate demands smaller RB block size, and smaller RB block size results in more buffered connection in the system, which also increases the overall system area. This effect can also be seen in the area curve in figure 20. This figure shows that for $P_f < 10^{-9}$ the total area is dominated by the logic replication factor which is the minimum possible area

overhead. The area curve follows the replication curve closely. For these fault rates, we also see a very small drop in the system throughput (figure 19). For higher fault rates the RB block size is reduced to prevent throughput loss. Reducing the rollback block size, however, results in more streaming interconnects in the system. Therefore the buffered connections start to consume a larger fraction of the total area. This fact causes the divergence of the total area overhead curve from the replication factor curve around $P_f = 10^{-9}$.

Figure 21 plots the area/throughput ratio for rollback recovery and feed-forward recovery techniques. As you can see our rollback technique not only reduces the area overhead by up to a factor of six, but from an area–time product point of view it is also a more efficient design.

7. Summary

Reliability techniques, such as *feed-forward recovery*, rely only on spatial redundancy. These techniques require large area overhead as the device failure rate increases. Here we developed and analyzed a recovery technique, *fine-grained rollback recovery*, that exploits redundancy in time as well as space. This technique has lower area overhead with negligible impact on performance for fault rates as high as $P_f = 10^{-10}$. At $P_f = 10^{-9}$ the replication factor is almost an order of magnitude smaller in rollback recovery than feed-forward recovery. For $P_f \leqslant 10^{-9}$, even the total area overhead of rollback can be about six times smaller than the feed-forward replication factor—and consequently much smaller than the complete area overhead required for a feed-forward implementation. At these fault rates, we show that detection is best performed using fine-grained detection blocks using 88 Pterms to protect 16 logical Pterms and rollback is best performed on larger blocks containing 450 K Pterms to protect 56 K logical Pterms.

Although the replication factor of rollback recovery remains relatively low for high fault rates, the total area overhead becomes large due to the streaming buffers. At high fault rates buffer area is the dominant area in streaming design. For example, for $P_f = 10^{-7}$, the buffered connection takes almost 2/3 of the total area. Therefore techniques which reduce this buffer overhead could offer even greater area benefit.

We used replication with comparison as the error detection technique because it has compact encoder and decoder circuits and allows general-purpose analysis. The total area overhead may be further reduced by using design-specific techniques, where applicable, that avoid simple replication of logic and checkers to provide efficient detection.

Acknowledgments

Heather Quinn and an anonymous reviewer provided valuable feedback on early versions of this article which helped improve the clarity and presentation.

This research was funded in part by National Science Foundation Grant CCF-0403674 and the Defense Advanced Research Projects Agency under ONR contract N00014-01-0651. This material is based upon work supported by the Department of the Navy, Office of Naval Research. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

References

[1] Hareland S, Maiz J, Alavi M, Mistry K, Walsta S and Dai C 2001 Impact of CMOS process scaling and SOI on the soft error rates of logic processes *Proc. Symp. on VLSI Digest of Technology Papers* pp 73–4

[2] Wakerly J 1978 *Error Detecting Codes Self-Checking Circuits and Applications* (Amsterdam: Elsevier–North-Holland)

[3] Nikolić K, Sadek A and Forshaw M 2002 Fault-tolerant techniques for nanocomputers *Nanotechnology* **13** 357–62

[4] Roy S and Beiu V 2004 Majority multiplexing–economical redundant fault-tolerant design for nanoarchitectures *IEEE Trans. Nanotechnol.* **4** 441–51

[5] Von Neumann J 1956 Probabilistic logic and the synthesis of reliable organisms from unreliable components *Automata Studies* ed C Shannon and J McCarthy (Princeton, NJ: Princeton University Press)

[6] DeHon A 2005 Design of programmable interconnect for sublithographic programmable logic arrays *Proc. Int. Symp. on Field-Programmable Gate Arrays (Feb.)* pp 127–37

[7] Kim J and Kish L 2004 Error rate in current-controlled logic processors with shot noise *Fluctuat. Noise Lett.* **4** 83–6

[8] International Technology Roadmap for Semiconductors 2005 http://www.itrs.net/Links/2005ITRS/Home2005.htm

[9] Swanson J A 1960 Physical versus logical coupling in memory systems *IBM J. Res. Dev.* **4** 305–10

[10] Stein K-U 1977 Noise-induced error rates as limiting factor for energy per operation in digital ic's *IEEE J. Solid State Circuits* **12** 527–30

[11] Austin T, Blaauw D, Mudge T and Flautner K 2004 Making typical silicon matter with razor *IEEE Comput.* **37** 57–65

[12] Kish L B 2002 End of Moores law: thermal (noise) death of integration in micro and nano electronics *Phys. Lett.* A **305** 144–9

[13] Morales A M and Lieber C M 1998 A laser ablation method for synthesis of crystalline semiconductor nanowires *Science* **279** 208–11

[14] Cui Y, Lauhon L J, Gudiksen M S, Wang J and Lieber C M 2001 Diameter-controlled synthesis of single crystal silicon nanowires *Appl. Phys. Lett.* **78** 2214–6

[15] Gudiksen M S, Wang J and Lieber C M 2001 Synthetic control of the diameter and length of semiconductor nanowires *J. Phys. Chem.* B **105** 4062–4

[16] Zheng B, Wu Y, Yang P and Liu J 2002 Synthesis of ultra-long and highly-oriented silicon oxide nanowires from alloy liquid *Adv. Mater.* **14** 122

[17] Wu Y, Xiang J, Yang C, Lu W and Lieber C M 2004 Single-crystal metallic nanowires and metal/semiconductor nanowire heterostructures *Nature* **430** 61–4

[18] DeHon A 2005 Nanowire-based programmable architectures *ACM J. Emerg. Technol. Comput. Syst.* **1** 109–62

[19] Collier C, Mattersteig G, Wong E, Luo Y, Beverly K, Sampaio J, Raymo F, Stoddart J and Heath J 2000 A [2] catenane-based solid state reconfigurable switch *Science* **289** 1172–5

[20] Gudiksen M S, Lauhon L J, Wang J, Smith D C and Lieber C M 2002 Growth of nanowire superlattice structures for nanoscale photonics and electronics *Nature* **415** 617–20

[21] DeHon A, Lincoln P and Savage J 2003 Stochastic assembly of sublithographic nanoscale interfaces *IEEE Trans. Nanotechnol.* **2** 165–74
[22] Yang C, Zhong Z and Lieber C M 2005 Encoding electronic properties by synthesis of axial modulation-doped silicon nanowires *Science* **310** 1304–7
[23] Bowen N S and Pradham D K 1993 Processor- and memory-based checkpoint and rollback recovery *Computer* **26** 22–31
[24] Chiu G-M and Young C-R 1996 Efficient rollback-recovery technique in distributed computing systems *IEEE Trans. Parallel Distrib. Syst.* **7** 565–77
[25] Pradhan D K and Vaidya N H 1997 Roll-forward and rollback recovery: performance-reliability trade-off *IEEE Trans. Comput.* **46** 372–8
[26] Kahn G 1974 The semantics of a simple language for parallel programming *Proc. IFIP Congr. 74* (Amsterdam: North-Holland) pp 471–5
[27] Lee E A and Messerschmitt D G 1987 Synchronous data flow *Proc. IEEE* **75** 1235–45
[28] Bove V M Jr and Watlington J A 1995 Cheops: A reconfigurable data-flow system for video processing *IEEE Trans. Cir. Syst. Video Technol.* **5** 140–9
[29] Shaw M and Garlan D 1996 *Software Architecture: Perspectives on an Emerging Discipline* (Englewood Cliffs, NJ: Prentice-Hall)
[30] de Kock E A, Essink G, Smits W J M, van der Wolf P, Brunel J-Y, Kruijtzer W M, Lieverse P and Vissers K A 2000 Application modeling for signal processing systems *Proc. Conf. on ACM/IEEE Design Automation* pp 402–5
[31] DeHon A, Markovsky Y, Caspi E, Chu M, Huang R, Perissakis S, Pozzi L, Yeh J and Wawrzynek J 2006 Stream computations organized for reconfigurable execution *J. Microprocessors Microsyst.* **30** 334–54
[32] Mitra S and McCluskey E J 2000 Which concurrent error detection scheme to choose? *Proc. Int. Test Conf.* pp 985–94
[33] Tuba N A and McCluskey E J 1997 Logic synthesis of multilevel circuits with concurrent error detection *IEEE Trans. Comput.-Aided Des. Integr. Circuit Syst.* **16** (7)
[34] Pradhan D K and Stiffler J J 1980 Error-correcting codes and self-checking circuits *Computer* **13** 27–37
[35] Lyons R E and Vandekulk W 1962 The use of triple-modular redundancy to improve computer reliability *IBM J. Res. Dev.* **6** 200
[36] Avizienis A 1973 Arithmetic algorithms for error-coded operands *IEEE Trans. Comput.* **C-22** 567–72
[37] Nicolaidis M 1993 Efficient implementation of self-checking adders and alus *Proc. 23rd Fault-Tolerant Computing Symp.* pp 586–95
[38] Nicolaidis M, Duarte R, Manich S and Figueras J 1997 Achieving fault secureness in parity prediction arithmetic operators *IEEE Des. Test Comput.* April–June
[39] Huang W-J, Saxena N R and McCluskey E J 2000 A reliable lz data compressor on reconfigurable coprocessors *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines* pp 249–58
[40] Naeimi H and DeHon A 2007 Fault secure encoder and decoder for memory applications *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (Sept.)*
[41] Jha N K and Wang S 1993 Design and synthesis of self-checking VLSI circuit *IEEE Trans. Comput.-Aided Des. Integr. Circuit Syst.* **12** 878–87
[42] Lala P K 2001 *Self-Checking and Fault-Tolerant Digital Design* (San Mateo, CA: Morgan Kaufmann)
[43] DeHon A and Naeimi H 2005 Seven strategies for tolerating highly defective fabrication *IEEE Des. Test Comput.* **22** 306–15
[44] DeHon A and Wilson M J 2004 Nanowire-based sublithographic programmable logic arrays *Proc. Int. Symp. on Field-Programmable Gate Arrays (Feb.)* pp 123–32
[45] Shivakumar P, Kistler M, Keckler S W, Burger D and Alvisi L 2002 Modeling the effect of technology trends on the soft error rate of combinational logic *Proc. Int. Conf. on Dependable Systems and Networks (June)* pp 389–98
[46] Naeimi H and DeHon A 2004 A greedy algorithm for tolerating defective crosspoints in NanoPLA design *Proc. Int. Conf. on Field-Programmable Technology (Dec.)* IEEE pp 49–56
[47] Betz V and Rose Jonathan 1999 FPGA Place-and-Route Challenge http://www.eecg.toronto.edu/vaughn/challenge/challenge.html
[48] Landman B S and Russo R L 1971 On pin versus block relationship for partitions of logic circuits *IEEE Trans. Comput.* **20** 1469–79